

Mem^p: Exploring Agent Procedural Memory

Runnan Fang^{♦♡*}, Yuan Liang^{♦*}, Xiaobin Wang[♡], Jialong Wu[♡],
 Shuofei Qiao^{♦♡}, Pengjun Xie[♡], Fei Huang[♡], Huajun Chen[♦], Ningyu Zhang^{♦†}

[♦]Zhejiang University [♡]Alibaba Group

{rolnan, zhangningyu}@zju.edu.cn

Abstract

Large Language Models (LLMs) based agents excel at diverse tasks, yet they suffer from brittle procedural memory that is manually engineered or entangled in static parameters. In this work, we investigate strategies to endow agents with a *learnable*, *updatable*, and *lifelong* procedural memory. We propose *Mem^p* that distills past agent trajectories into both fine-grained, step-by-step instructions and higher-level, script-like abstractions, and explore the impact of different strategies for *Build*, *Retrieval*, and *Update* of procedural memory. Coupled with a dynamic regimen that continuously updates, corrects, and deprecates its contents, this repository evolves in lockstep with new experience. Empirical evaluation on TravelPlanner and ALFWorld shows that as the memory repository is refined, agents achieve steadily higher success rates and greater efficiency on analogous tasks. Moreover, procedural memory built from a stronger model retains its value: migrating the procedural memory to a weaker model yields substantial performance gains.

Introduction

As large language models (LLMs) grow ever more powerful, LLM-based agents augmented by their own reasoning and external tools are taking on increasingly sophisticated works (Zhao et al. 2023; Wang et al. 2024a; Xi et al. 2025; Qiao et al. 2023). No longer mere assistants, these agents now trawl the web for elusive insights and weave them into comprehensive, publication ready reports, like Deep Research (OpenAI 2025; x.ai 2025) and Web-Dancer (Wu et al. 2025a). Moreover, they can handle complex data analyses (Lan et al. 2025; Ifargan et al. 2025; Ou et al. 2025), navigate multi-step GUI workflows (Luo et al. 2025; Qin et al. 2025), and sustain long-horizon, tool-rich interactions (Yao et al. 2025; Barres et al. 2025; Chen et al. 2025; Fang et al. 2025; Gur et al. 2023) with precision. Yet executing such intricate, long-horizon tasks demands dozens of steps and protracted runtimes. Along the way, unpredictable external events—network glitches, UI changes, shifting data schemas—can derail the entire process. Restarting from scratch every time is a punishing ordeal for present-day agents. Beneath their surface diversity, many complex tasks share deep structural commonalities

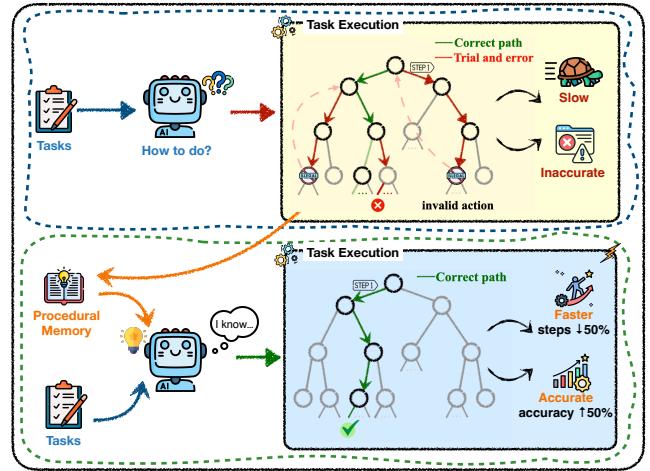


Figure 1: With **procedural memory**, agents can improve both the success rate (accuracy \uparrow) and execution efficiency (steps \downarrow) when solving similar tasks.

and a similar environment. Instead of starting fresh each time, an agent should extract its experience from past successes. By turning earlier trajectories into reusable templates like patterns of reasoning, tool sequences, and recovery tactics, it can progress step by step, learning from every failure and success, until even the most convoluted missions become routine.

The capacity to distill, chronicle, and re-apply lessons from one's own experiential trajectory is the bedrock of human learning and the pivotal gateway through which an agent ascends toward self-directed refinement (Liu et al. 2025a; Sumers et al. 2023a; Li et al. 2023). Procedural memory (Gupta and Cohen 2002; Cohen and Squire 1980) silently compiles habitual skills into executable subroutines, enabling unconscious, fluent action. While contemporary agents built on LLMs can compose short action plans or call external tools, their procedural knowledge is either hand-crafted, stored as brittle prompt templates, or implicitly entangled in model parameters that are expensive to update. Existing memory-augmented frameworks such as LangGraph (Mavroudis 2024), AutoGPT (Yang, Yue, and He 2023), or agent cognitive architectures like

* Equal Core Contributors.

† Corresponding Author.

Memory Bank (Zhong et al. 2024a; Sumers et al. 2023b) and Soar (Laird 2022) provide coarse abstractions (buffers, rule chunks, production systems) but leave the optimization of procedural memory life-cycle operations about how skills are built, indexed, patched, and eventually pruned, largely unexamined. Consequently, there is no principled way to quantify how efficiently an agent evolves its procedural repertoire or to guarantee that new experiences improve rather than erode performance.

To close this gap, we present Mem^P , a task-agnostic framework that treats procedural memory as a first-class optimization object. The core exploration of Mem^P lies in how different strategies for memory construction, retrieval, and updating affect overall performance. During the construction phase, we follow the majority of traditional memory architectures and agent-based memory designs by leveraging either the full historical trajectory or explicit guidelines to guide the process. In the retrieval phase, we experiment with various key-building strategies—such as query-vector matching and keyword-vector matching—to investigate how procedural memory can be constructed more precisely. Unlike prior memory mechanisms or learning from experience, Mem^P introduces diverse procedural-memory update strategies: In the realm of agents, memory updating is crucial for agents to adapt to dynamic environments. By incorporating diverse strategies like ordinary addition, validation filtering, reflection, and dynamic discarding, agents can efficiently manage their knowledge base. This ensures they stay updated with new information, discard outdated data, and optimize memory resources. Such strategies enhance learning efficiency, improve decision-making quality, and boost adaptability, allowing agents to perform optimally in various tasks and scenarios.

We instantiate Mem^P on top of strong LLMs (GPT-4o and Claude, Qwen) and evaluate on two diverse domains: long-horizon housework ALFWorld (Shridhar et al. 2021) and long-term information seeking task TravelPlanner (Xie et al. 2024). On two benchmark datasets that rigorously evaluate agent capabilities, we demonstrate that constructing and retrieving procedural memory during training empowers an agent to distill and reuse its prior experience. When this memory is exploited at test time, the agent’s task accuracy rises, and compared with tackling each instance in isolation, it eliminates most fruitless exploration on unfamiliar tasks, yielding substantial reductions in both step count and token consumption. Further, by equipping the agent with a set of memory-update mechanisms, we allow it to build and refine its procedural memory while acting in the test environment. This endows the agent with a continual, almost linear mastery of the task. Extensive ablations reveal that procedural memory also scales gracefully and transfers effectively to new, related tasks.

Related Works

Memory in Language Agents. Memory is a foundational component in language agents, enabling them to retain and utilize past information across multiple timescales, including short-term, episodic, and long-term memory, to enhance their performance and adaptability (Zhou et al. 2023,

2024; Zhang et al. 2024; Liu et al. 2025a; Li et al. 2025). These systems aim to mimic aspects of human memory to improve coherence, personalization, and learning capabilities (Chhikara et al. 2025; Wu et al. 2025b; Xia et al. 2025). Current approaches include end-to-end memory systems (Yu et al. 2025; Zhou et al. 2025), external memory systems (Chhikara et al. 2025; Zhong et al. 2024b), and hierarchical memory structures (Hu et al. 2024a; Xu et al. 2025). These methods involve encoding and storing information in various formats, using retrieval mechanisms like vector embeddings and semantic search, and implementing memory updating and forgetting strategies to maintain relevance and efficiency. Despite its importance, memory in multi-turn agent interactions remains underexplored, and enabling agents to effectively learn and utilize memory across trajectories poses a significant challenge. Procedural memory is a type of long-term memory that involves the retention of procedures and skills, such as typing or riding a bike, which are performed automatically without conscious thought. The agent utilizes procedural memory to internalize and automate repetitive tasks, decision-making processes, and interaction patterns, leading to more efficient and context-aware responses over time. Although there have been several works, such as Voyager (Wang et al. 2023), AWM (Wang et al. 2024b), and AutoManual (Chen et al. 2024), that utilize procedural memory to enhance agents’ capabilities on similar tasks, there still lacks a systematic analysis on how to construct, retrieve, and update such procedural memory like (Wu et al. 2024). Therefore, our work mainly focuses on exploring how to build an effective procedural memory system for agents performing cross-trajectory tasks.

Learning from Experience. LLM-based Agent learning from experience involves intelligence continuously improving their decision-making capabilities through interaction with environments and utilization of past experiences (Tan et al. 2025; Tang et al. 2025; Zhou et al. 2025; Qiao et al. 2025; Su et al. 2025; Wang et al. 2024b). This approach is crucial for developing adaptive and intelligent agents capable of handling dynamic real-world scenarios, as it allows them to optimize behaviors, reduce manual programming needs, and enhance performance across various tasks (Zheng et al.; Liu et al. 2025c; Wang et al. 2025). Agents typically employ mechanisms such as reinforcement learning (Lu et al. 2025; Dong et al. 2025), experience replay (Feng et al. 2025; Liu et al. 2025b), imitation learning (Sun et al. 2024; Yang et al. 2024b), memory management (Hou, Tamoto, and Miyashita 2024; Hu et al. 2024b), and multi-agent learning to achieve this. However, current methods face limitations including low sample efficiency, poor generalization across tasks, catastrophic forgetting when learning new information, and there are very few features for memory update. The key distinction of our work lies in systematically investigating optimal strategies for construction, retrieval, and update modules of an agent’s procedural knowledge. During the update phase, we enhance the agent’s capabilities by maintaining an editable repository of procedural knowledge. Additionally, collecting high-quality training data can be challenging and may introduce biases. Addressing these limi-

tations is essential for advancing the capabilities of LLM-based agents and ensuring their effective application in real-world contexts.

Preliminary

When an agent influences its external environment by invoking external tools or executing prescribed actions, and iteratively refines its behavior over multiple rounds to accomplish a complex multi-step objective, this paradigm can be modeled as a Markov Decision Process (MDP). (Puterman 1990) Under this view, at each discrete time step t , the agent, situated in state $s_t \in S$, chooses an action $a_t \in A$, according to its policy $\pi(a_t|s_t)$, where A is the action space of the task. The environment then transitions to a new state $s_{t+1} \in S$ and emits an observation O_t . Consequently, the entire interaction trajectory may be compactly expressed as:

$$\tau = (s_0, a_0, o_1, s_1, a_1, o_2, \dots, s_T), \quad (1)$$

where τ is the complete exploration trajectory of this task. Moreover, a reward function R will evaluate the task's completion r within this environment env by assigning a score based on the final state s_T or the entire trajectory τ .

$$r = R(env, s_T, \tau) \in [0, 1] \quad (2)$$

Although approaches resembling Markov Decision Processes inevitably contain erroneous actions and exploratory attempts, the contextual information they generate becomes valuable for decision-making as the model's reasoning and reflective capabilities improve. Nevertheless, this benefit comes at a high test-time cost—both in time and in token consumption. When facing an entirely new and complex environment, many actions (or tokens) are spent simply understanding the environment and the task itself. This leads to redundancy when similar tasks are executed within the same environment: the agent has already acquired partial procedural knowledge about the environment or task during earlier episodes, yet fails to transfer that knowledge effectively to subsequent tasks.

By shifting from parallel task completion to sequential task completion, the agent can learn and distill experience from earlier tasks, thereby reducing repetitive exploration. Inspired by human procedural memory, we propose to equip the agent with a procedural memory module. This module transforms the conventional policy $\pi(a_t|s_t)$ into $\pi_{m^p}(a_t|s_t)$, where m^p is the agent's learned procedural memory.

Agent Procedural Memory

Procedural memory is the type of long-term memory responsible for knowing how to perform tasks and skills, such as typing or riding a bike. By mastering this type of procedural memory, humans avoid the need to relearn the process each time. For an agent, that is, for a task trajectory τ and its reward r , a memory m^p is constructed by a builder B , thereby achieving the acquisition of memory, namely

$$Mem = \sum_{t=1}^T m^{p_t}, \text{ where } m^{p_t} = B(\tau_t, r_t) \quad (3)$$

where Mem is the procedural memory library acquired by the agent over the T tasks. After constructing the procedural memory library, when facing a new task t_{new} , we need a good procedural memory retriever to recall a memory that fits t_{new} . Generally speaking, we would choose the task $t \in T$ that is most similar to t_{new} , because similar experiences are more helpful for the agent to complete the new task.

$$m_{retrieved} = \arg \max_{m^{p_i} \in Mem} S(t_{new}, t_i) \quad (4)$$

As we use cosine similarity for the vector embedding model ϕ of the task in the experiment, the retrieval process becomes:

$$m_{retrieved} = \arg \max_{m^{p_i} \in Mem} \frac{\phi(t_{new}) \cdot \phi(t_i)}{\|\phi(t_{new})\| \|\phi(t_i)\|}. \quad (5)$$

Moreover, as the number of completed tasks continuously increases, simply augmenting the agent's procedural memory is inconsistent with common sense. A well-designed procedural memory system should have a reasonable update mechanism—that is, it should dynamically perform addition, deletion, modification, and retrieval based on the task execution context.

Let $M(t)$ denote the agent's procedural memory at time t , and τ_t represent the set of tasks completed up to time t . Then, the update mechanism can be modeled as a function U that takes the current procedural memory and task execution feedback to produce the updated memory:

$$M(t+1) = U(M(t), E(t), \tau_t), \quad (6)$$

where $E(t)$ encapsulates the execution feedback (e.g., success, failure, performance metrics). A more sophisticated implementation of U could be represented as:

$$U = Add(M_{new}) \ominus Remove(M_{obso}) \oplus Update(M_{exist}), \quad (7)$$

where M_{new} represents new procedural memory to be added; M_{obso} indicates procedural memory to be removed, $M_{existing}$ are tasks to be updated based on execution feedback $E(t)$. This comprehensive formula captures the essential add, delete, and modify operations within the update mechanism.

Experiment

In this section, we will introduce the Procedural Memory framework in detail (Figure 2), covering the storage, retrieval, and update modules of memory, as well as analyzing which strategies perform better within each module.

Experimental Settings

Datasets. For our experiments, we adapt TravelPlanner (Xie et al. 2024) and ALFWorld (Shridhar et al. 2021) benchmarks. TravelPlanner is a benchmark designed to evaluate agents' ability to use tools and perform complex planning under intricate constraints. In contrast, ALFWorld comprises household tasks. In each interaction round, the agent outputs an action, and the environment responds with textual feedback describing the resulting state. This process repeats for multiple turns until the task is completed or the maximum number of rounds is reached. ALFWorld includes test split to evaluate the agent's generalization ability.

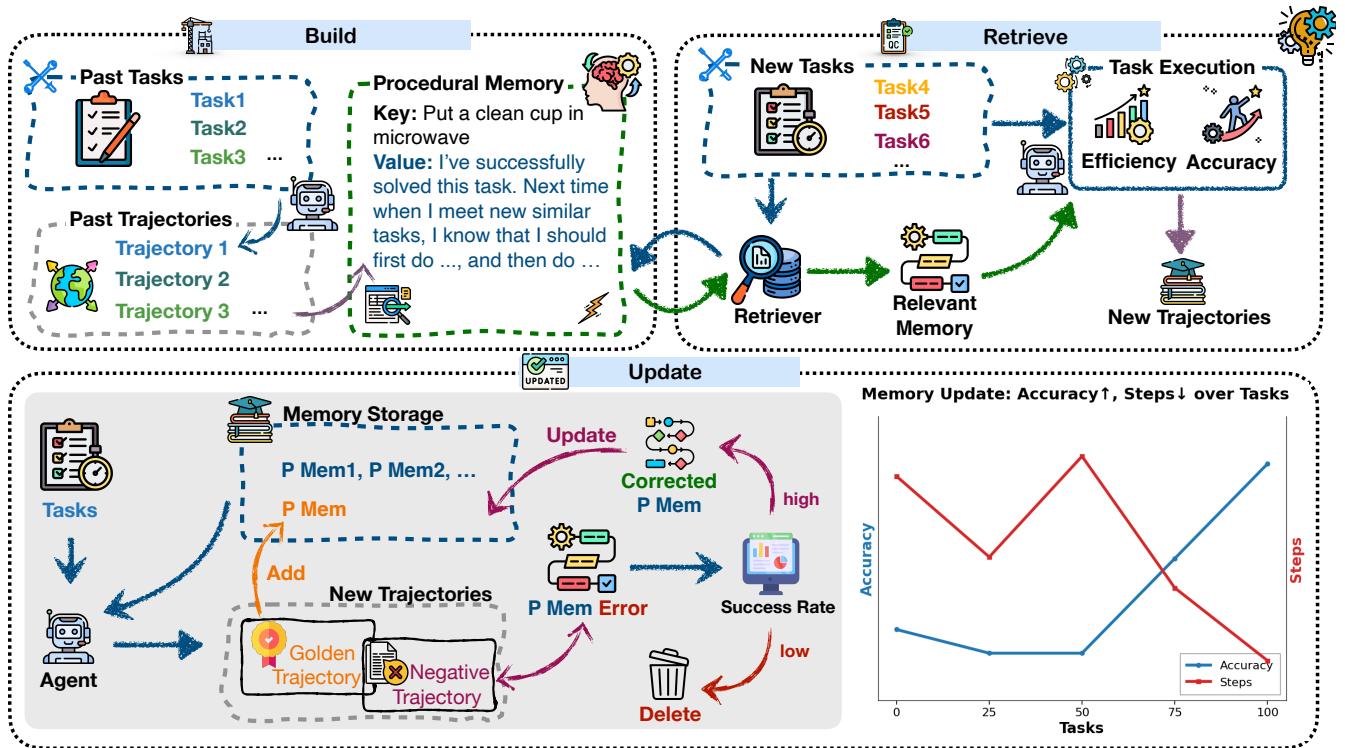


Figure 2: The procedural memory framework consists of **Build**, **Retrieve**, and **Update**, which respectively involve encoding stored procedural memory, forming new procedural memories, and modifying existing ones in light of new experiences.

Backbones. In our experiments, we benchmarked our procedural memory on three base models. Specifically, we adopt the two proprietary frontier models that have consistently dominated public leaderboards: OpenAI’s GPT-4o (OpenAI 2022) and Anthropic’s Claude (Anthropic 2022), and complement them with the open-sourced Qwen2.5-72B-Instruct (Yang et al. 2024a). The first two provide state-of-the-art closed-source performance, while the third allows us to verify that our findings generalize beyond proprietary systems and remain valid in the open-source regime.

Evaluation. For ALFWORLD dataset, task completion is evaluated by the execution environment. After a task is completed or the maximum number of execution steps is reached, the environment provides a reward of 0 or 1 to indicate whether the task has been successfully completed. For TravelPlanner, we conduct experiments on the test set in a two-stage mode. After multiple rounds of interaction to obtain the travel trajectory and the final planner, GPT-4o converts the travel plan into a specified JSON format. The converted plan is then compared with the gold standard to obtain scores for both Common Sense and Hard Constraint.

Memory Storage & Retrieval

Procedural knowledge is typically stored in two main formats: (1) trajectories are kept verbatim, round by round, in memory, or (2) high-level abstractions are extracted from these trajectories and then stored. Once a similar procedu-

ral memory is retrieved, it is appended to the task as part of the context, serving as prior knowledge to assist the model in completing the task.

Inspired by this, we designed the following experimental conditions:

- **No Memory:** The model tackles the assigned task in a ReAct fashion without any external memory.
- **Trajectory:** We first filter the gold trajectories from the training set and store them. At inference time, the system retrieves the top-k trajectories whose query vectors are most similar to the current task’s vector, supplying them as procedural memories before execution.
- **Script:** The model analyzes and summarizes the gold trajectories from the training set, distilling them into abstract procedural knowledge that is provided as a prompt before each task.
- **Proceduralization:** This condition combines the full retrieved trajectories with the high-level script generated by the model, integrating both concrete examples and abstract guidance as the procedural memory.

As shown in Table 1, all memory construction methods outperform the no-memory baseline, achieving higher scores on both datasets while also reducing the number of steps required. This indicates that procedural memory built during training is beneficial for directly applying tasks during testing. Furthermore, we observe that the approach of abstracting trajectories into scripts during training yields rel-

Model	Granularity	TravelPlanner			ALFWorld		
		#CS ↑	#HC ↑	Steps ↓	Dev ↑	Test ↑	Steps ↓
GPT-4o	No Memory	71.93	12.88	17.84	39.28	42.14	23.76
	Script	72.08	5.50	<u>15.79</u>	<u>66.67</u>	56.43	18.52
	Trajectory	<u>76.02</u>	8.25	<u>14.64</u>	67.17	74.29	<u>16.49</u>
	Proceduralization	79.94	9.76	14.62	87.14	77.86	15.01
Claude-3.5-sonnet	No Memory	63.49	33.06	18.84	39.20	34.97	24.12
	Script	62.08	29.61	19.21	<u>56.13</u>	53.59	19.38
	Trajectory	<u>65.76</u>	29.61	<u>17.72</u>	<u>69.28</u>	71.78	<u>15.97</u>
	Proceduralization	65.46	30.14	15.29	82.50	74.72	15.79
Qwen2.5-72b	No Memory	56.57	7.34	18.32	44.91	41.25	21.38
	Script	58.59	7.34	<u>18.53</u>	<u>66.24</u>	61.88	17.13
	Trajectory	<u>63.41</u>	<u>12.66</u>	<u>18.12</u>	<u>64.49</u>	<u>69.57</u>	<u>16.40</u>
	Proceduralization	63.82	14.19	17.94	85.71	77.19	15.32

Table 1: Results on **Build Policy**. #CS, #HC denote Commensense and Hard Constraint, respectively. ↑ indicates the higher values are better, and ↓ denotes the lower values are better. The best results among all methods with similar settings are **bolded**, and the second-best results are underlined.

atively better performance on the ALFWorld test set compared to the dev set. Conversely, trajectories that utilize complete execution traces as procedural memory achieve higher scores on the dev set, suggesting that scripts are more capable of generalizing to different test tasks, while trajectories are better suited for scenarios involving tasks similar to those already completed. By combining procedure knowledge from both methods of employing abstracted guidelines along with concrete execution trajectories, we attain the optimal performance.

After converting a set of completed trajectories into procedural memory, the next critical challenge is to retrieve the most accurate and relevant procedural knowledge when a new task arrives. We have designed several different key construction methods for memory storage to facilitate subsequent vector-based matching and retrieval:

- **Random Sample:** Does not utilize keys for vector retrieval; instead, randomly extracts a few memories from procedural memory.
- **Query:** Employ query description as the key for storage, leveraging the semantic similarity of queries for retrieval.
- **AveFact:** We apply a large model to extract keywords from the task’s query, then computes the average similarity across matched keywords for retrieval.

During the retrieval process, we evaluate the similarity by calculating the cosine similarity between their corresponding vectors. Our experiments show that these different retrieval strategies produce varying results. Specifically, compared to random sampling, employing the query based and AveFact methods for precise retrieval significantly improves performance. The query-based approach benefits from capturing semantic contexts, enabling more accurate matches. The AveFact method, by extracting key features and averaging their similarities, effectively focuses on core task elements, leading to better retrieval efficacy. Overall, our findings suggest that incorporating semantic understanding and key feature extraction in retrieval strategies substantially enhances memory access accuracy and the effectiveness of downstream task performance.

Model	Policy	#CS ↑	#HC ↑	Steps ↓
GPT-4o	No Memory	71.93	12.88	17.84
	Random Sample	<u>74.59</u>	6.72	<u>15.12</u>
	Key=Query	<u>73.38</u>	<u>8.95</u>	15.44
	Key=AveFact	76.02	8.25	14.64
Claude-3.5-sonnet	No Memory	63.49	33.06	18.84
	Random Sample	63.99	<u>29.91</u>	17.93
	Key=Query	<u>64.93</u>	28.56	17.60
	Key=AveFact	65.76	29.61	<u>17.72</u>
Qwen2.5-72b	No Memory	56.57	7.34	18.32
	Random Sample	59.76	8.43	<u>18.31</u>
	Key=Query	<u>61.71</u>	<u>11.97</u>	18.54
	Key=AveFact	63.41	12.66	18.12

Table 2: Results on **Retrieve Policy** on TravelPlanner.

Memory Update

While many prior efforts have focused on developing reusable procedural knowledge, enabling models to learn from prior experiences rather than solving each test task in isolation, most existing memory update methods remain quite rudimentary. Typically, they simply append newly acquired memories to the existing store—a so-called “merge” strategy. In this work, we explore several online memory-update mechanisms to identify which dynamic strategy delivers the best performance on our tasks. Beyond end-to-end evaluation metrics, we also analyze how both accuracy and efficiency evolve as the number of executed tasks increases, explicitly measuring the benefits conferred by our procedural memory.

To facilitate systematic comparison, we designed several memory-update scenarios. In each, the agent’s episodic memory is refreshed after every t test-set tasks. The specific update strategies are as follows:

- **Vanilla Memory Update:** After every t tasks, all trajectories from these tasks are consolidated into procedural memories and directly appended to the memory bank.
- **Validation:** After every t tasks, only the trajectories of

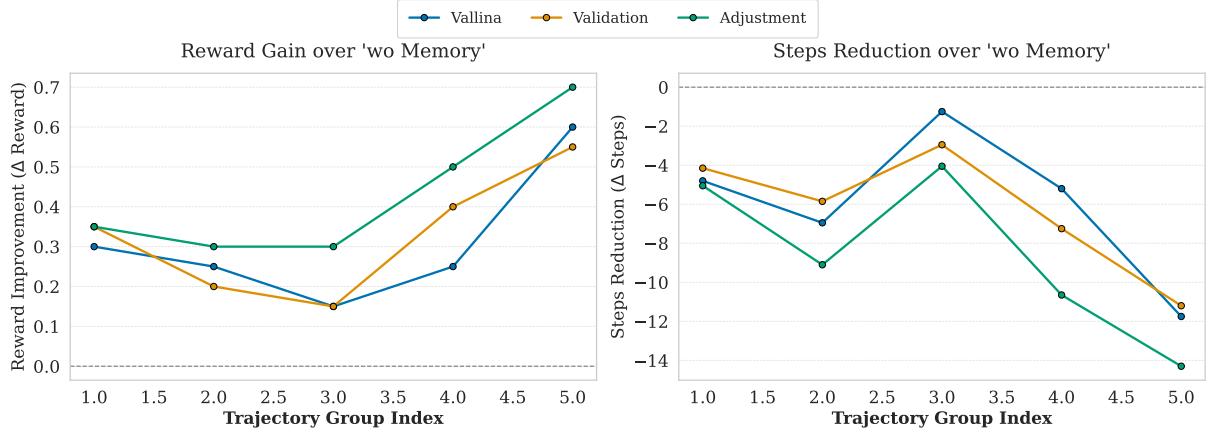


Figure 3: Reward gain and steps reduction vs. trajectory group index with **procedural memory**.

successfully completed tasks are retained and converted into procedural memories for storage.

- **Adjustment:** When a retrieved procedural memory results in a failed execution, the erroneous trajectory is combined with the original memory and then revised in place, yielding an updated procedural memory.

As depicted in Figure 3, we systematically divided the tasks within our testbed into several distinct groups, with each group comprising a diverse set of individual tasks. Upon the completion of tasks within each group, we employed the previously described strategies to construct, store, and update the procedural memory. The experimental results reveal a clear trend: as we sequentially progress through more groups and iteratively refresh the memory, all strategies contribute to improved performance on subsequent tasks. Specifically, this is reflected not only in higher overall scores but also in a reduction in the number of steps required to complete the tasks.

A closer comparison of different strategies exposes significant disparities in their effectiveness. Notably, the reflexion-based update mechanism stands out as the most effective approach. By the time the final group of tasks is reached, this method delivers a substantial advantage: it surpasses the second-best strategy by an impressive margin of +0.7 points and achieves a reduction of 14 steps. These improvements underscore the value of continually updating the memory, particularly when the update is guided by an error-correction mechanism embedded in the reflexion process.

Analysis

Procedural Memory Boosts Accuracy and Cuts Trials. Figure 5 presents a case study demonstrating how Procedural Memory enhances both accuracy and efficiency. In the absence of Procedural Memory, facing a complex task that has not been performed before, there are usually two situations. In the first scenario (left), the model repeatedly attempts illegal or incorrect actions, causing the context to become increasingly complex and eventually exceeding the model’s understanding capacity. In the second scenario

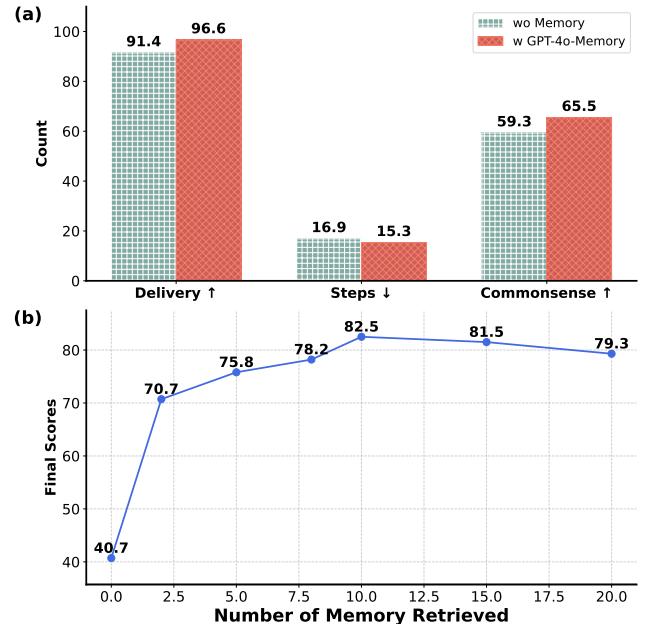


Figure 4: (a) Transfer result of GPT-4o’s procedural memory to Qwen2.5-14B-Instruct and its performance on TravelPlanner dataset.(b) The relationship between the quantity of procedural memory retrieved for GPT-4o’s performance on the ALFWorld dataset.

(middle), after multiple attempts, the model completes the task but at a cost significantly higher than the optimal path.

In contrast, once Procedural Memory is available for similar tasks, the model spends less time on trial and error. For example, in the egg heating problem, Procedural Memory can indicate the approximate location of the egg, saving the aimless search. During the heating process, it provides clear guidance, ensuring that the heating actions are performed consecutively and correctly, thereby allowing the task to be completed in fewer steps.

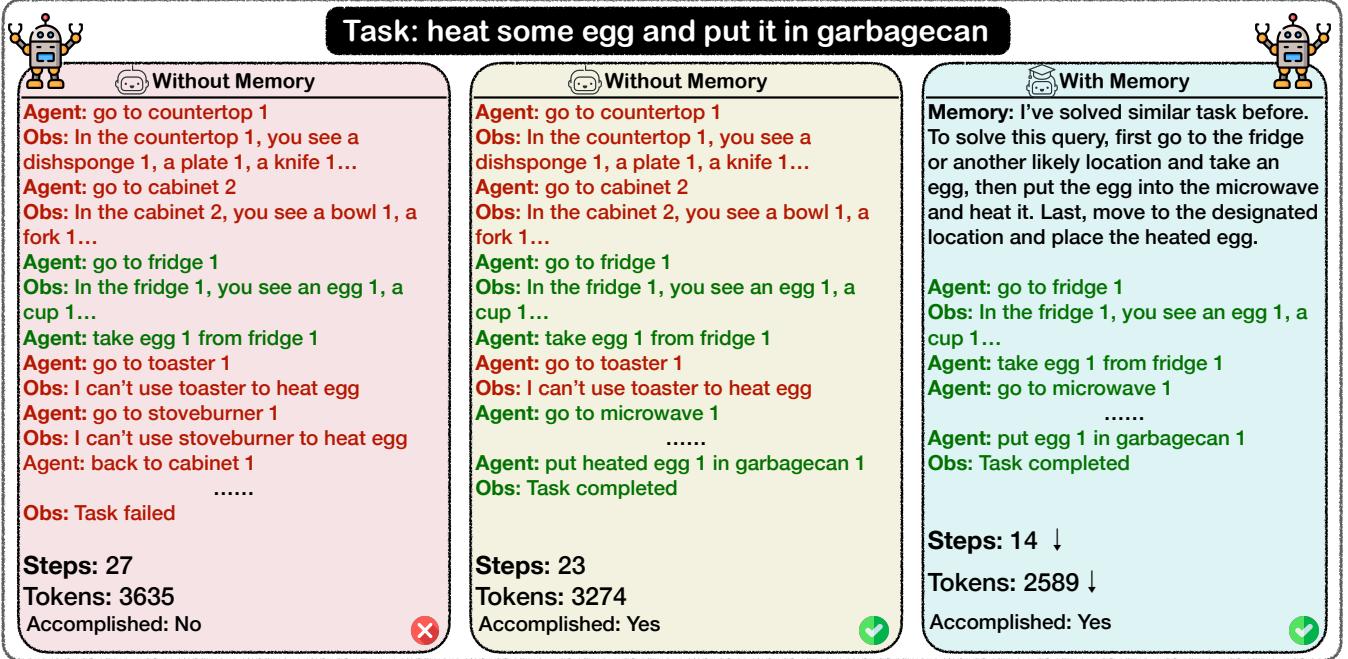


Figure 5: Compare trajectories with and without procedural memory, shortens the process by **9** steps and saves **685** tokens.

Procedural memory exhibits transferability from strong models to weaker ones. For a procedural memory constructed from a strong model in an offline memory library, we aim to verify whether this form of procedural memory can be effectively transferred to other models, or even weaker models. This exploration underscores the significance of memory transfer, as it could potentially enhance the adaptability and efficiency of various models by leveraging the knowledge and experience encapsulated within the strong model’s memory structure. As shown in Figure 4 (b), procedural memory generated by GPT-4o was employed by Qwen2.5-14B. On the Travel Plan benchmark, the 14 billion parameter model raised its task completion rate by 5% and cut the average number of steps by 1.6. Similar gains, both in success rate and trajectory length, appeared on ALF-World. These outcomes confirm that procedural knowledge from a stronger model can be distilled into a reusable memory bank and transferred to a smaller model with minimal overhead, giving that smaller model a clear boost in task solving ability. Moreover, by leveraging procedural memory transfer, we can rapidly migrate the experiential knowledge that one model has acquired to another, which is highly beneficial for agents as they adapt to new tasks with greater efficiency and robustness.

Scaling Memory Retrieval Improves Agent Performance. While our main experiment has already demonstrated that procedural memory improves an agent’s task accuracy and reduces the number of steps required, vector-based storage and retrieval confer an advantage over human procedural memory: they can be scaled both in total capacity and in the number of memories retrieved. To investigate whether an agent’s performance continues to rise

as the procedural-memory store and the number of retrieved memories increase, we designed a set of follow-up experiments. As shown in Figure 4 (b), as the number of retrieved procedural memories increases, the agent’s performance also improves steadily, exhibiting an upward trend followed by a plateau. However, retrieving too many memories can lead to a decline in the agent’s performance. This is because excessive retrieval can affect the context length and also introduce less accurate procedural memories, which can interfere with the overall effectiveness.

Conclusion and Future Work

We introduce Mem^p , a task-agnostic framework that elevates procedural memory to a core optimization target in LLM-based agents. By systematically studying strategies for memory construction, retrieval, and updating, Mem^p enables agents to distill, reuse, and refine their own past experiences across diverse, long-horizon tasks. Empirical results on housework automation and information-seeking benchmarks show that leveraging procedural memory significantly boosts task success rates and efficiency. Beyond improving individual episodes, Mem^p supports continual learning and robust generalization, marking a step toward self-improving, resilient agents.

In our experiments, Mem^p has achieved promising results in both construction and retrieval. Moving forward, we plan to enhance this work in several ways. Firstly, we will develop more diverse retrieval strategies. The current approach involves constructing different keys for vector-based retrieval. However, traditional methods like BM25 could also be explored to retrieve precise memories more effectively. Secondly, in Mem^p , we currently rely on the standard

reward signals provided by the benchmark. However, in real-world scenarios, many tasks do not have clear reward signals, making it difficult for the agent to determine whether a task has been completed successfully. In such cases, using a large language model (LLM) as a judge to assess task completion could be a viable solution. This would transform the agent's lifecycle into a continuous loop of executing tasks, self-assessing completion, building memories, and then proceeding to new tasks.

References

- Anthropic. 2022. Claude 3.5 Sonnet System Card.
- Barres, V.; Dong, H.; Ray, S.; Si, X.; and Narasimhan, K. 2025. τ^2 -Bench: Evaluating Conversational Agents in a Dual-Control Environment.
- Chen, C.; Hao, X.; Liu, W.; Huang, X.; Zeng, X.; Yu, S.; Li, D.; Wang, S.; Gan, W.; Huang, Y.; et al. 2025. ACEBench: Who Wins the Match Point in Tool Usage?
- Chen, M.; Li, Y.; Yang, Y.; Yu, S.; Lin, B.; and He, X. 2024. AutoManual: Constructing Instruction Manuals by LLM Agents via Interactive Environmental Learning. arXiv:2405.16247.
- Chhikara, P.; Khant, D.; Aryan, S.; Singh, T.; and Yadav, D. 2025. Mem0: Building Production-Ready AI Agents with Scalable Long-Term Memory.
- Cohen, N. J.; and Squire, L. R. 1980. Preserved learning and retention of pattern-analyzing skill in amnesia: Dissociation of knowing how and knowing that.
- Dong, G.; Chen, Y.; Li, X.; Jin, J.; Qian, H.; Zhu, Y.; Mao, H.; Zhou, G.; Dou, Z.; and Wen, J.-R. 2025. Tool-Star: Empowering LLM-Brained Multi-Tool Reasoner via Reinforcement Learning.
- Fang, R.; Wang, X.; Liang, Y.; Qiao, S.; Wu, J.; Xi, Z.; Zhang, N.; Jiang, Y.; Xie, P.; Huang, F.; et al. 2025. SynWorld: Virtual Scenario Synthesis for Agentic Action Knowledge Refinement.
- Feng, E.; Zhou, W.; Liu, Z.; Chen, L.; Dong, Y.; Zhang, C.; Zhao, Y.; Du, D.; Hua, Z.; Xia, Y.; et al. 2025. Get Experience from Practice: LLM Agents with Record & Replay.
- Gupta, P.; and Cohen, N. J. 2002. Theoretical and computational analysis of skill learning, repetition priming, and procedural memory.
- Gur, I.; Furuta, H.; Huang, A.; Safdari, M.; Matsuo, Y.; Eck, D.; and Faust, A. 2023. A real-world webagent with planning, long context understanding, and program synthesis.
- Hou, Y.; Tamoto, H.; and Miyashita, H. 2024. "my agent understands me better": Integrating dynamic human-like memory recall and consolidation in llm-based agents. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 1–7.
- Hu, M.; Chen, T.; Chen, Q.; Mu, Y.; Shao, W.; and Luo, P. 2024a. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model.
- Hu, M.; Chen, T.; Chen, Q.; Mu, Y.; Shao, W.; and Luo, P. 2024b. Hiagent: Hierarchical working memory management for solving long-horizon agent tasks with large language model.
- Ifargan, T.; Hafner, L.; Kern, M.; Alcalay, O.; and Kishony, R. 2025. Autonomous llm-driven research—from data to human-verifiable research papers.
- Laird, J. E. 2022. Introduction to the soar cognitive architecture.
- Lan, W.; Tang, Z.; Liu, M.; Chen, Q.; Peng, W.; Chen, Y. P.; and Pan, Y. 2025. The large language models on biomedical data analysis: a survey.
- Li, G.; Hammoud, H.; Itani, H.; Khizbulin, D.; and Ghanem, B. 2023. Camel: Communicative agents for "mind" exploration of large language model society.
- Li, Z.; Song, S.; Xi, C.; Wang, H.; Tang, C.; Niu, S.; Chen, D.; Yang, J.; Li, C.; Yu, Q.; et al. 2025. Memos: A memory os for ai system.
- Liu, B.; Li, X.; Zhang, J.; Wang, J.; He, T.; Hong, S.; Liu, H.; Zhang, S.; Song, K.; Zhu, K.; et al. 2025a. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems.
- Liu, Y.; Si, C.; Narasimhan, K.; and Yao, S. 2025b. Contextual Experience Replay for Self-Improvement of Language Agents.
- Liu, Z.; Chai, J.; Zhu, X.; Tang, S.; Ye, R.; Zhang, B.; Bai, L.; and Chen, S. 2025c. Ml-agent: Reinforcing llm agents for autonomous machine learning engineering.
- Lu, F.; Zhong, Z.; Liu, S.; Fu, C.-W.; and Jia, J. 2025. ARPO: End-to-End Policy Optimization for GUI Agents with Experience Replay.
- Luo, R.; Wang, L.; He, W.; and Xia, X. 2025. Gui-r1: A generalist r1-style vision-language action model for gui agents.
- Mavroudis, V. 2024. LangChain v0. 3.
- OpenAI. 2022. GPT-4 System Card.
- OpenAI. 2025. Deep Research System Card.
- Ou, Y.; Luo, Y.; Zheng, J.; Wei, L.; Qiao, S.; Zhang, J.; Zheng, D.; Chen, H.; and Zhang, N. 2025. AutoMind: Adaptive Knowledgeable Agent for Automated Data Science.
- Puterman, M. L. 1990. Markov decision processes.
- Qiao, S.; Fang, R.; Qiu, Z.; Wang, X.; Zhang, N.; Jiang, Y.; Xie, P.; Huang, F.; and Chen, H. 2025. Benchmarking Agentic Workflow Generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.
- Qiao, S.; Ou, Y.; Zhang, N.; Chen, X.; Yao, Y.; Deng, S.; Tan, C.; Huang, F.; and Chen, H. 2023. Reasoning with Language Model Prompting: A Survey. In Rogers, A.; Boyd-Graber, J. L.; and Okazaki, N., eds., *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2023, Toronto, Canada, July 9-14, 2023, 5368–5393. Association for Computational Linguistics.
- Qin, Y.; Ye, Y.; Fang, J.; Wang, H.; Liang, S.; Tian, S.; Zhang, J.; Li, J.; Li, Y.; Huang, S.; et al. 2025. Ui-tars: Pioneering automated gui interaction with native agents.

- Shridhar, M.; Yuan, X.; Côté, M.; Bisk, Y.; Trischler, A.; and Hausknecht, M. J. 2021. ALFWorld: Aligning Text and Embodied Environments for Interactive Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Su, H.; Sun, R.; Yoon, J.; Yin, P.; Yu, T.; and Arik, S. Ö. 2025. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments.
- Sumers, T.; Yao, S.; Narasimhan, K.; and Griffiths, T. 2023a. Cognitive architectures for language agents.
- Sumers, T.; Yao, S.; Narasimhan, K.; and Griffiths, T. 2023b. Cognitive architectures for language agents.
- Sun, J.; Zhang, Q.; Duan, Y.; Jiang, X.; Cheng, C.; and Xu, R. 2024. Prompt, plan, perform: Llm-based humanoid control via quantized imitation learning. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 16236–16242. IEEE.
- Tan, X.; Li, B.; Qiu, X.; Qu, C.; Chu, W.; Xu, Y.; and Qi, Y. 2025. Meta-Agent-Workflow: Streamlining Tool Usage in LLMs through Workflow Construction, Retrieval, and Refinement. In *Companion Proceedings of the ACM on Web Conference 2025, WWW '25*, 458–467. New York, NY, USA: Association for Computing Machinery. ISBN 9798400713316.
- Tang, X.; Qin, T.; Peng, T.; Zhou, Z.; Shao, D.; Du, T.; Wei, X.; Xia, P.; Wu, F.; Zhu, H.; Zhang, G.; Liu, J.; Wang, X.; Hong, S.; Wu, C.; Cheng, H.; Wang, C.; and Zhou, W. 2025. Agent KB: Leveraging Cross-Domain Experience for Agentic Problem Solving. arXiv:2507.06229.
- Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An open-ended embodied agent with large language models.
- Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; et al. 2024a. A survey on large language model based autonomous agents.
- Wang, Z.; Xu, H.; Wang, J.; Zhang, X.; Yan, M.; Zhang, J.; Huang, F.; and Ji, H. 2025. Mobile-agent-e: Self-evolving mobile assistant for complex tasks.
- Wang, Z. Z.; Mao, J.; Fried, D.; and Neubig, G. 2024b. Agent workflow memory.
- Wu, J.; Li, B.; Fang, R.; Yin, W.; Zhang, L.; Tao, Z.; Zhang, D.; Xi, Z.; Fu, G.; Jiang, Y.; et al. 2025a. WebDancer: Towards Autonomous Information Seeking Agency.
- Wu, J.; Yin, W.; Jiang, Y.; Wang, Z.; Xi, Z.; Fang, R.; Zhang, L.; He, Y.; Zhou, D.; Xie, P.; and Huang, F. 2025b. Web-Walker: Benchmarking LLMs in Web Traversal. In Che, W.; Nabende, J.; Shutova, E.; and Pilehvar, M. T., eds., *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 10290–10305. Vienna, Austria: Association for Computational Linguistics. ISBN 979-8-89176-251-0.
- Wu, X.; Bu, Y.; Cai, Y.; and Wang, T. 2024. Updating Large Language Models' Memories with Time Constraints.
- x.ai. 2025. Grok 3 Beta — The Age of Reasoning Agents.
- Xi, Z.; Chen, W.; Guo, X.; He, W.; Ding, Y.; Hong, B.; Zhang, M.; Wang, J.; Jin, S.; Zhou, E.; et al. 2025. The rise and potential of large language model based agents: A survey.
- Xia, M.; Ruehle, V.; Rajmohan, S.; and Shokri, R. 2025. Minerva: A Programmable Memory Test Benchmark for Language Models.
- Xie, J.; Zhang, K.; Chen, J.; Zhu, T.; Lou, R.; Tian, Y.; Xiao, Y.; and Su, Y. 2024. TravelPlanner: A Benchmark for Real-World Planning with Language Agents. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.
- Xu, W.; Liang, Z.; Mei, K.; Gao, H.; Tan, J.; and Zhang, Y. 2025. A-mem: Agentic memory for llm agents.
- Yang, A.; Yang, B.; Zhang, B.; Hui, B.; Zheng, B.; Yu, B.; Li, C.; Liu, D.; Huang, F.; Wei, H.; et al. 2024a. Qwen2. 5 technical report.
- Yang, H.; Yue, S.; and He, Y. 2023. Auto-gpt for online decision making: Benchmarks and additional opinions.
- Yang, Y.; Zhou, T.; Li, K.; Tao, D.; Li, L.; Shen, L.; He, X.; Jiang, J.; and Shi, Y. 2024b. Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 26275–26285.
- Yao, S.; Shinn, N.; Razavi, P.; and Narasimhan, K. R. 2025. τ -bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. In *The Thirteenth International Conference on Learning Representations*.
- Yu, H.; Chen, T.; Feng, J.; Chen, J.; Dai, W.; Yu, Q.; Zhang, Y.-Q.; Ma, W.-Y.; Liu, J.; Wang, M.; and Zhou, H. 2025. MemAgent: Reshaping Long-Context LLM with Multi-Conv RL-based Memory Agent. arXiv:2507.02259.
- Zhang, Z.; Bo, X.; Ma, C.; Li, R.; Chen, X.; Dai, Q.; Zhu, J.; Dong, Z.; and Wen, J.-R. 2024. A survey on the memory mechanism of large language model based agents.
- Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models.
- Zheng, L.; Wang, R.; Wang, X.; and An, B. ???? Synapse: Trajectory-as-Exemplar Prompting with Memory for Computer Control. In *The Twelfth International Conference on Learning Representations*.
- Zhong, W.; Guo, L.; Gao, Q.; Ye, H.; and Wang, Y. 2024a. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 19724–19731.
- Zhong, W.; Guo, L.; Gao, Q.; Ye, H.; and Wang, Y. 2024b. Memorybank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 19724–19731.
- Zhou, W.; Jiang, Y. E.; Li, L.; Wu, J.; Wang, T.; Qiu, S.; Zhang, J.; Chen, J.; Wu, R.; Wang, S.; Zhu, S.; Chen, J.; Zhang, W.; Tang, X.; Zhang, N.; Chen, H.; Cui, P.; and Sachan, M. 2023. Agents: An Open-source Framework for Autonomous Language Agents. arXiv:2309.07870.

Zhou, W.; Ou, Y.; Ding, S.; Li, L.; Wu, J.; Wang, T.; Chen, J.; Wang, S.; Xu, X.; Zhang, N.; Chen, H.; and Jiang, Y. E. 2024. Symbolic Learning Enables Self-Evolving Agents. arXiv:2406.18532.

Zhou, Z.; Qu, A.; Wu, Z.; Kim, S.; Prakash, A.; Rus, D.; Zhao, J.; Low, B. K. H.; and Liang, P. P. 2025. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents. arXiv:2506.15841.